Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 637–653
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

A model of design decision making based on empirical results of interviews with software designers

Carmen Zannier [a,*], Mike Chiasson [b], Frank Maurer [a]

[a] University of Calgary, Department of Computer Science, Calgary, AB, Canada
[b] University of Lancaster, Department of Management Science, Management School, Lancaster University, Bailrigg, Lancaster, UK

Available online 13 February 2007

Abstract

Despite the impact of design decisions on software design, we have little understanding about how design decisions are made. This hinders our ability to provide design metrics, processes and training that support inherent design work. By interviewing 25 software designers and using content analysis and explanation building as our analysis technique, we provide qualitative and quantitative results that highlight aspects of rational and naturalistic decision making in software design. Our qualitative multi-case study results in a model of design decision making to answer the question: how do software designers make design decisions? We find the structure of the design problem determines the aspects of rational and naturalistic decision making used. The more structured the design decision, the less a designer considers options.

Keywords: Design decision; Rational decision making; Naturalistic decision making; Interviewing

1. Introduction

Designing software involves numerous cognitive skills such as mental modeling [25], mental simulation [1], problem structuring [24] and decision making [27], the last of which is the focus of this paper. We empirically examine software design decisions for three reasons. First, there is little empirical work in this area despite strong calls to examine this topic [1,15,27,28,54,65]. Second, growing support for studying the social side of software development makes an empirical examination of design decisions highly relevant to research in software design, especially due to growing recognition that "the major problems of [software design] work are not so much technological as sociological in nature" [16]. Last, an empirically based understanding of design decisions motivates the development of design processes, tools and metrics that incorporate inherent work activities of software design. Given this, we describe our multi-case study of 25 software designers interviewed about design decisions they have made, and we present quantitative and qualitative results of these interviews and the decision model that emerged from our results.

We provide two key conclusions. First, our results are consistent with [24], that design is primarily about problem structuring. We add to this work by showing that problem structuring in software design is a crucial aspect of the design decision making process. Our second conclusion is that designers utilize two seemingly opposing decision making approaches, concurrently: Rational decision making (RDM) [41], and Naturalistic decision making (NDM) [37]. Rational decision making is characterized by the consequential choice of an option among a set of options, with a goal of selecting the optimal option [41]. Naturalistic decision making is characterized by situation assessment and the evaluation of a single option with a goal of selecting a satisfactory option [37]. The current state of decision making literature suggests these approaches are independent of each other. For example, fire-fighters use NDM [37], and operations researchers

* Corresponding author. Tel.: +1 403 771 6633.
E-mail addresses: zannierc@cpsc.ucalgary.ca (C. Zannier), m.chiasson@lancaster.ac.uk (M. Chiasson), maurer@cpsc.ucalgary.ca (F. Maurer).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.02.010

use RDM [41]. Our results show that in software design, decisions are made using aspects of NDM and aspects of RDM concurrently, contingent upon the structure of the problem, as defined by the decision maker.

These results impact software design in three ways: design metrics, design processes and design training. Regarding design metrics, our results challenge the pursuit of only objective and quantifiable measures of design, via numerous qualitative descriptions of design (e.g. "verbose", "awkward", "spaghetti") and no quantitative descriptions. Regarding design processes, our results support iterative development approaches not only because it provides a designer with more opportunities to structure information surrounding a decision, but because iterative development accommodates our empirical evidence that designers often follow singular evaluation not consequential choice among design alternatives – an NDM perspective. Iterative development provides the opportunity to revisit a design decision that was made via singular evaluation, and then evaluate alternatives as needed. Lastly, in design training our results raise numerous questions about the most effective methods to acquire design knowledge, given that some designers pursue knowledge in places such as textbooks, conferences or the internet, while other designers show minimal interest in such knowledge.

Two assumptions were made that are relevant for this paper. The first was that a design change involves one or more design decisions. The second was that the design change is a critical incident, as per the definition [19].

Our paper proceeds as follows. Section 2 provides a literature review and Section 3 introduces the study, describing data collection and analysis. Section 4 provides our results, including the decision model and Section 5 discusses validity. Section 6 concludes this research. [68] provides definitions of codes used in content analysis, referenced throughout this paper.

2. Literature review

We describe four topics pertinent to a review of software design decision making and presented in increasing order of relevance to this research. The topics are listed in Table 1, in the leftmost column. A common theme emerged from an examination of these topics: that software design emerges from two *seemingly* opposing perspectives. For example, rational decision making is one perspective from which to view decision making; naturalistic decision making is the other perspective. It can be claimed that the definition of naturalistic decision making opposes the definition of rational decision making.

Table 1
Pertinent literature review topics

Topic	Perspective	Perspective
Problem solving	Well structured problems	Ill-structured problems
Measuring software design	Objective	Subjective
Software design cognition	Explicit	Implicit
Decision making	Rational	Natural

Avoiding a right and wrong response to rational and naturalistic decision making, we show how the two complement each other in addressing the topic. *We suggest that RDM and/or NDM provides one platform from which to identify and discuss tensions and complementarities in decision making approaches.* Among the topics there is similarity in the ideas inherent in the perspectives. For example, the ideas found in the perspective called well structured are used by the ideas found in the perspective called explicit. The format of Table 1 suggests similarity among the perspectives listed in the same column, and differences between the columns. Despite the obvious differences between the columns, our goal of this section is to explore these as complementary differences instead of radical and irreconcilable approaches.

2.1. Problem solving

The extent to which a problem is structured establishes two perspectives from which to view a problem. These perspectives are a *well structured problem* (WSP) and an *ill structured problem* (ISP). They can be viewed by considering problems in general (e.g. the design of a house, a game of chess) or by considering software development problems specifically.

A well structured problem is one that has criteria that reveal relationships between the characteristics of a problem domain and the characteristics of a method by which to solve the problem [58]. "There is at least one problem space in which can be represented the initial problem state, the goal state, and all other states that may be reached, or considered, in the course of attempting a solution to a problem" [58]. An example of a well structured problem is a game of chess.

An ill-structured problem is a problem that is not well-structured [58]. An ill structured problem requires problem structuring, where a problem solver converts an ill structured problem to a well structured problem [58]. A problem solver spends more time in problem structuring than in actually solving a problem once it is structured [58]. An example of an ill structured problem is the design of a new house [58]. Decisions made in early design sketches of the house establish structures under which following decisions will be made [58].

Specific to software development, a well structured problem can be handled by Tayloristic approaches. Taylorism is a management solution, where factories are managed through scientific methods rather than by "the rule of thumb". Scientific selection of the workperson, task breakdown and separating planning from execution are three characteristics of this approach [61]. Taylorism is a

potential foundation for software engineering practice which is often considered to be *opposed* to agile practices [2].

Specific to software development, an ill structured problem provides a platform for wicked problems. The formulation of a wicked problem is the actual problem; wicked problems have no stopping rule; solutions to wicked problems are not true or false, but good or bad [53]. Wicked problems are a potential foundation for software engineering practices that *agree with* agile practices [2,52].

The dichotomy between well structured problems and ill structured problems is easily reconcilable. "There is no real boundary between well structured problems and ill structured problems." [58]. Given the definitions of the two, much of problem solving can be viewed as problem structuring [58].

2.2. Measuring software design

When measuring software design the literature shows two perspectives from which to begin. They are quantifiable metrics and qualifiable metrics.

Quantifiable software design metrics must accurately represent the attributes they are supposed to quantify [33]. Consequently, much work teaches the proper use of metrics [18]. It is even stated that software engineering cannot become a true engineering disciplines, until the field establishes proper measurement theories [18]. In practice, consistent relationships between internal and external software design attributes are exceedingly difficult to develop [9]. This is because software does not directly show design attributes but only exhibits characteristics indicative of design attributes [9]. This is also because "good" software design is inconsistently defined [10]. Dating back at least 25 years, researchers have tried to develop a consistent and agreed upon set of design metrics, to no avail.

Qualifiable software design metrics relies on more elusive concepts such as quality or goodness, and rely on human judgment of "good" [17,5]. Recognizing that "good" software design is undefined, [21], qualifiable software design uses the concept of "good enough" software (satisificing), to evaluate a software design. Additionally, qualifiable software design metrics rely on design patterns as the cornerstone of good software design [22]. Software design patterns provide common solutions to common design problems but still require a designer to tailor a design pattern to a specific design solution [22].

Reconciling quantifiable and qualifiable software design metrics is difficult because it is akin to matching internal product characteristics to external design attributes. As of yet, this has not been accomplished.

2.3. Capturing (Design) cognition

Capturing software design cognition (i.e. the way a software designer thinks) can be done explicitly or implicitly and much research exists in both areas. The use of design rationale approaches is an explicit approach to capturing design cognition. Empirically studying software designers at work is an implicit approach to capturing design cognition.

2.3.1. Explicit capture: Design rationale

Design rationale is the documenting of design decisions and their justifications [39,35]. The purpose is to facilitate reasoning and communication about a design as well as organizational learning [39,35]. Two examples of design rationale capture are *Issue-Based Information System* (IBIS) [13] and *Questions, Options and Criteria* (QOC) [42]. Unfortunately, evaluations of design rationale have not been positive. An initial investigation attempted to determine the usefulness of design rationale documents [31]. The conclusion of the study was that design rationale documents should be useful to interested designers. However, design rationale documents were, at the time, insufficient, as less than half of the designers' design rationale questions were answered by the design rationale documentation. A second study investigated the usability of QOC [55]. This study critiqued QOC as lacking a vocabulary, providing a poor representation of dependencies between decision problems, and being too restrictive [55]. Lastly, speaking to the motivation behind design rationale, one investigation found software design meetings to be structured and was thus critical of the usefulness of design rationale models as a method to organize design [47].

2.3.2. Implicit capture: Design studies

A survey of design studies reveals the following six related qualities impact software design: expertise, mental modeling, mental simulation, continual restructuring, preferred evaluation criteria and group interactions. Each quality is defined below, and the results of the studies pertaining to each quality are summarized.

Expertise is the knowledge and experience software designers have in design [1,25,4,59]. The issue of expertise, or knowledge, is fundamental to software productivity and quality [15]. "Although individual staff members understood different components of the application, the deep integration of various knowledge domains required to integrate the design of a large, complex system was a scarcer attribute" [15]. Individual team members do not possess enough knowledge for a project and must learn additional information before they are productive [65].

Mental modeling is the internal or external model a designer creates that is capable of supporting mental simulation [1]. "A model is an abstract representation of a system that enables us to answer questions about the system" [11]. External modeling is the development of diagrams, code or any physical model of the application [4]. Internal modeling is the mental development of a model of the application [24]. Studies have shown that expert designers can create more detailed mental models than novices can, and their mental models are better able to handle mental

simulations [1,15,24,59]. In addition, individuals maintain a mental model that is used in the creation of a group model [23]. A model develops as team members learn from one another about the application and required computational structures [65].

Mental simulation is the "ability to imagine people and objects consciously and to transform those people and objects through several transitions, finally picturing them in a different way than at the start" [37]. External and internal modeling provides a foundation upon which mental simulations run. Designers alternate between concrete scenarios and abstracting from these scenarios [23]. Mental simulations run on a mental model and are thus subject to the detail of the mental model as well as limitations of working memory [1,15,24,25,32].

Continual restructuring is the process of turning an ISP (ill-structured software problem) to a WSP (well-structured software problem). Continual restructuring emphasizes the importance of the context of a situation in evaluating its execution. The current state of design determines relevant issues in a design problem [4,20,43] and designers revisit the framed problem as new context-based information emerges [23,65]. This process has been termed "organized anarchy" [12].

The term "*preferred evaluation criteria*" refers to the minimal criteria a subject adopts to structure an ISP and guide the search for a satisfactory solution [24]. Caution must be taken in the selection of preferred evaluation criteria. Properly chosen criteria reduced design problem complexity, but poorly chosen criteria led to early reduction of the design problem and a closed minded approach to the situation [24,25,65]. The preferred evaluation criteria generated can be considered similar to that of confirmation bias, tunnel vision and groupthink [64,46].

Group interactions is the dynamics of group work in software design. Motivation for the examination of group interactions arises from vast amounts of time designers spend in informal communication [26]. The terms "distributed" and "shared" cognition suggest that individual mental models coalesce via coordinated group action, resulting in a common model [23]. The coalescing may result from the domination of a small coalition of individual designers (or sometimes just one designer) controlling the direction of the project [15]. In relation to the preferred evaluation criteria, one study showed this control can result in success or failure of a closed-minded group upon the arrival of new design ideas [65].

Reconciling explicit design cognition capture and implicit design cognition capture, is not well researched in software development, perhaps because it is akin to evaluating how people work versus how they say they work. Such research is beyond the scope of this work.

2.4. Decision making

Two approaches to decision making provide the perspectives from which to view multiple decision making theories as well as the perspectives from which to view software design decision making. The two decision making approaches differ in their emphasis on mathematical foundations and real-time scenarios. They are called rational decision making and naturalistic decision making.

2.4.1. Rational decision making

A rational decision, "is one that conforms either to a set of general principles that govern preferences or to a set of rules that govern behaviour. These principles or rules are then applied in a logical way to the situation of concern resulting in actions which generate consequences that are deemed to be acceptable to the decision maker" [60].

Rational decision making is characterized by an appreciation for mathematical computation of decision alternatives. A rational decision is also a normative decision, because the process and outcome are prescriptive, often in opposition to description. A normative decision "prescribes for given assumptions, courses of action for the attainment of outcomes having certain formal 'optimum' properties" [41]. "Prescriptive" is defined as recommending an approach to decision making to the decision maker.

A rational decision has three features that are parts of the decision maker's approach to decision making [57,56].

- *Decision alternatives*. The decision alternatives are represented by a set of possible courses of action and potential outcomes for each action.
- *Utility function*. A utility function assigns a value to each possible action based on its outcome.
- *Probabilities*. A decision has information or probabilities as to which outcome will occur in case of the selection of a particular alternative.

Rational decision making is limited by three assumptions. The first is that a set of possible courses of action and the probability of outcomes is actually known. The second is that the decision maker's goal is to optimize. The last assumption is that combinatorial explosion of alternatives and the time involved in mathematical calculations of situations of combinatorial explosion are not a large concern [48,37].

2.4.2. Naturalistic decision making

A naturalistic decision, "connotes situational behaviour without the conscious analytical division of situations into parts and evaluation according to context-independent rules" [20]. Naturalistic decision making is characterized by an appreciation for real-time scenarios and judgment biases. A naturalistic decision is defined by the following six characteristics [36]:

- Manifests itself in dynamic and continually changing conditions.
- Embodies real-time reactions to these changes.
- Embraces ill-defined tasks and goals.
- Resolves itself under the guidance of knowledgeable decision makers.

- Utilizes situation assessment over consequential choice.
- Has a goal of satisficing instead of optimizing.

Three terms in these six characteristics require definition: consequential choice, situation assessment and satisficing.

- *Consequential choice*. The organized analysis of options and potential outcomes, typical of rational decision theory [40].
- *Situation assessment*. The absence of choice in real-world decisions. A decision maker exercises an action that may or may not be considered amongst a set of choices [40]. A decision maker thinks of a possible option then uses mental simulation to evaluate the option. If the mental simulation succeeds the decision maker executes that option.
- *Satisficing*. The acceptance of a satisfactory, as opposed to optimal, outcome [37].

Naturalistic decision theories are limited by the absence of a definable, predictable set of outcomes. Our review has found 11 approaches to naturalistic decision making [51,29,14,63,8,62,45,7,50,30,37].

3. The study

We accomplished our empirical examination of design decision making via a multi-case study that used interviews and content analysis [38] to understand design decision making. Our multi-case study consisted of 25 explanatory case studies with software designers [66]. Our interviews followed a semi-structured format and lasted, on average, 45 min. We used content analysis to code words, phrases, sentences and paragraphs. Lastly we used our results to build an explanation of design decision making [66]. The purpose of this examination is to answer the question: how do software designers make design decisions? In [67] we presented hypotheses for design decisions made for four types of design decisions. Given our emphasis on problem structuring in our current results we cannot group decisions into one of these four types, and we leave this categorization for future work.

3.1. Data collection

There are three issues we address from our interviewing, which was conducted from May to August of 2004. The interviews were audio recorded, totaling approximately $16\frac{1}{2}$ h of conversation.

To stimulate discussions around software decision making, we used the Critical Decision Method for Eliciting Knowledge (CDM) for our interview format [34]. The CDM begins with one general question to initiate conversation with the interview subject and hear a description of the critical incident. The CDM then provides probing questions based on the information given in this description. It is a semi-structured interview format [49]. In our interviews we asked designers to describe a design change they made, followed by probing questions addressing aspects of their description. The order and the way in which the questions were asked varied, depending upon the interview subject's description, but the theme of the question remained the same. The probing questions (e.g. Cues) and examples of the way the probing questions were asked, are listed in Table 2. An example of a variation of a question is, for external goals, "You mentioned your marketing department wanted you to use XSLT, can you talk about that a bit more?" We chose the CDM because of its extensive use in analyzing decisions [37] and we found it to be extremely beneficial because it generated conversation that we perceived to be natural for most interview subjects and ensured an approximately consistent approach to all interviews.

To collect data, we used snowball sampling, an emergent sampling approach that does not *explicitly* characterize an interview subject by domain, type of system, experience or any other characteristic. [49]. For our study, we initially interviewed colleagues at the University of Calgary who had previous software design experience and local colleagues (i.e. in the same city as us) we knew in the industry through conferences or user groups. This meant that initially our interview subjects were primarily Calgary-based software developers (11 participants). When a leading agile conference was held in Calgary in August 2004 [3], we broadened the demographic of our interview subjects. We interviewed leaders of the agile community and developers from across North America. We interviewed 25 people in total. During our interviewing, we noticed a difference in the perspectives of 8 of our interview subjects. They discussed design changes they had seen people make or had overseen, when they were a coach or a consultant on a team. They spoke about design change as a concept rather than one critical incident, and gave multiple examples of critical incidents of design changes. Given this, we defined two perspectives in our interviews subjects: Developer and

Table 2
Critical decision interview example questions [34]

Decision (initial question)	Describe how you make a design change to a system, and how you make the decision to make the change.
Cues (probe)	What do you see, hear, discuss, or experience that suggests a change needs to occur?
Knowledge (probe)	Where do you acquire the knowledge to make the change?
Options (probe)	Discuss the extent to which you consider options in making a design change to a system.
Experience (probe)	To what extent do specific past experiences impact your decision to make a design change?
Time pressure (probe)	How does time pressure impact decisions in design changes?
Externals (probe)	How do external goals impact decisions in design changes?

Mentor. A *Developer* is an interview subject who discussed a design change that they championed when s/he was a member of a design/development team, and discussed the design change as a critical incident. A *Mentor* is an interview subject who discussed design change as an abstract concept, based on the culmination of design experiences with software development teams where s/he was a coach or paid consultant. The abstract level at which Mentors discussed design changes allowed us to compare the general theories of design work to the actual practices of design. When we refer to all subjects of our interviews, Developers and Mentors, we say Designers. We recognize that 19 of our 25 Designers discussed design changes on Agile projects [2]. We address this when we discuss the validity of our study.

The interviews took place primarily at the time and location most convenient for the interview subject: at his/her place of business, over lunch or at a conference s/he was attending. We attempted to make the interview subject as comfortable as possible in this regard. Interview subjects explicitly stated when s/he did not remember enough about a critical incident to answer part of a question and also stated biases they might have had (although they did not use the word 'bias). Willingness to participate was not an issue for any of our 25 interviews. Lastly, we saw signs of a desire to be a good subject in 2–3 of our subjects. This was apparent when interview subjects said a version of the phrase "I'm not sure if that is the sort of answer you're looking for." To this we typically replied with a version of the phrase "I'm looking for anything you can recall surrounding the design change." In general though, we found the interview subjects had significant industry experience and/or were well-recognized members of their development community. They had little to gain or lose from participating, at any level, in our study.

3.2. Data analysis

We discuss three issues in our data analysis, which occurred on transcripts of the interviews. Transcripts of the recordings produced 180 typed pages (single space). The transcripts were marked with a pen colour and label indicating a code found during analysis [38]. Analysis was time consuming. For example, 1 h of recorded conversation took 12–15 h to analyze, depending upon the speed at which the interview subject spoke. This did not include time to compare results across interviews.

Our content analysis placed words, phrases, sentences and/or paragraphs into codes [38]. In our study we coded at two levels for our quantitative results. First we coded words or small phrases, which we called Small Window Coding (SWC). SWC removed the context from a transcript but showed word choice in the transcription. We then coded long phrases, sentences or paragraphs, which we called Large Window Coding (LWC). LWC incorporated more context of a statement than SWC and showed some word choice within a context. Given the large absence of context, we found SWC and LWC were most useful to generate frequencies of codes as an indicator of popular and unpopular codes. For SWC we used an interactive dictionary and for LWC we used that dictionary as a predefined dictionary, and added codes as needed.

We then generated further understanding of software decision making by showing relationships between codes in two ways. We used codes from SWC and LWC to show relationships between codes, throughout the transcripts. We called this Generic Relational Coding (GRC). We showed a relationship between two codes with a line. For example <<Ideas>> – <<Model>> was a common relationship showing the interview subject had a thought about the design of a system. GRC did not answer a specific interview question but did incorporate the context of a transcript and showed recurring themes throughout the transcripts. Next we used codes from SWC and LWC to show relationships between codes in answer to our interview questions. We called this Specific Relational Coding (SRC). SRC incorporated much context of an interview and answered a specific interview question.

To build explanations [66], we compared case study results to confirm and produce theory [66]. Using our four types of coding, we built a summary of each case study. Using our knowledge of decision making (RDM and NDM in particular), we generated small ideas and theories about each case study and compared them against other case studies. We continuously refined these ideas and theories and continuously incorporated more case studies, until our theory explained all our case studies. Our theory became our explanatory model of design decision making with the 25 case studies consistent with it.

Using six approaches to analyze our interview transcripts SWC, LWC, GRC, SRC, Case Study Summary and Cross Case Comparison, we were able to draw upon the strengths and avoid the limitations of some analysis forms with respect to our goal of understanding design decision making. Each form of analysis built our understanding of each case and was crucial in validating the overall result. While the analysis was lengthy, these six forms of analysis provided the most insight and value to our emerging theories.

4. Results

We provide five types of results before describing our decision model. First we provide frequencies of codes, which give us an idea of potentially important concepts to examine. Using this we then look at emerging themes of these concepts. We apply these themes to interpret how RDM and NDM map to design decision making, as our third type of results. These interpretations are used when we describe our case study summaries. Lastly the case study summaries are compared in our cross case analysis.

4.1. Frequencies

We examined the number of interview subjects who had a code in their top *three* codes. These are listed in Tables 3 and 4, for SWC and LWC respectively and show the most popular codes across interviews. We grouped the top *three* codes when we counted these frequencies because of how little variation there was in the frequencies of the top three codes (which we do not show, due to space constraints). Across all interviews, <<Modeling>> and <<Customer Product>> were in the top three codes more often than any other code in SWC. <<Modeling>> and <<Idea>> were in the top three codes more often than any other code in LWC. We find these numeric results show the dominant topics in the interviews but are insufficient indicators of design decision making and thus show codes that only had a frequency of 5 or more in the top three codes across interviews.

4.2. Emerging themes

Numerous relationships between codes emerged through our Generic Relational Coding, showing popular themes across the interviews. We discuss five here, given the three most popular codes presented in Tables 3 and 4.

4.2.1. <<Ideas>>

The first relationship that was popular was the relationship between <<Ideas>> and <<Better vs. Worse>>. Our interview subjects qualified their ideas according to some subjective definition of better or worse, as opposed to right or wrong. We provide the quote below as an example.

> "*The idea itself, I saw some of the custom attribute implementations and I thought that's a much better way of doing [what] Java did." Developer Perspective*

Table 3
Frequencies of Top 3 Codes for SWC across interviews

Code	Frequency
Modeling	13
Customer product	10
Group	9
Technology	9
Time	9
Frequency	7
Better vs. worse	5

Table 4
Frequencies of Top 3 Codes for LWC across interviews

Code	Frequency
Modeling	15
Idea	12
Customer product	7
Decision	7
Knowledge	7
Better vs. worse	6
Technology	6
Adaptation	5

This is important for design decisions because it highlights a difference between naturalistic decision making (NDM) and rational decision making (RDM). NDM utilizes a satisfactory solution while RDM utilizes an optimal solution. The relationship, <<Ideas>> – <<Better vs. Worse>> suggests designers think subjectively about design, which raises the question: is it possible to find optimality through individuals' subjective evaluations of right or wrong?

To be clear, the interview subjects discussed *design* ideas as being better or worse. This became apparent because the second dominant relationship across the interviews was <<Ideas>> – <<Modeling>> and the third was <<Modeling>> – <<Better vs. Worse>>. An example of the relationship between <<Ideas>> and <<Modeling>> is as follows.

> *"I was trying to go with [the] incremental change approach...um, where **did** I get the idea [for the design]? ...I guess it's kind of a basic encapsulation idea." Developer Perspective*

Through quotes such as these, we saw that designers had ideas about design that were sometimes difficult to trace back to a specific reference. The question raised above can now be narrowed to design: is it possible to find an optimal software design through designers' subjective evaluations of right or wrong software design? The example of the relationship between <<Modeling>> and <<Better vs. Worse>> reiterates this point.

> *"...during those 8 months we had to let certain things slide because [the system] just had to work, it didn't have to have...all the bells and whistles. It just really had to work." Developer Perspective*

4.2.2. <<Modeling>>

Another dominant relationship was one between <<Modeling>> and <<Adaptation>>. This relationship was prominent across the interviews, perhaps due to the agile background from which 19/25 of our subjects spoke. Nevertheless, changing design was important. An example of adapting a model is the following quote.

> *"...[There] was a heavily expensive object to create. In one part of the code we were creating it much more than was required. So the simple fix there was to create it once, and keep it there and just continue to reuse it." Developer Perspective*

Through quotes such as these, we saw that changing an existing design was discussed much more than beginning a new design. This gives merit to the idea that problem structuring builds on past structured problems, which we discuss in Section 4.4.1.

4.2.3. <<Customer Product>>

Lastly, a dominant relationship with <<Customer Product>> was <<Need>>. Customer requests were often

highlighted as a need (i.e. requirement) for the model. For example,

> *"There was no graphic at all before. It was presented in table form and [the customer] wanted to present the information graphically. That was definitely a customer need." Mentor Perspective.*

We found numerous relationships throughout GRC, but present these only because of the popularity of the codes <<Modeling>>, <<Ideas>> and <<Customer Product>>, as per Tables 3 and 4. There were two reasons to examine the recurring themes via GRC. The first was to give us an understanding of recurring themes in the transcripts and to allow these ideas to emerge with as little bias as possible. Ultimately, some of these recurring themes became empirical evidence that challenge traditional ideas about software design. For example, the above question about design optimality via subjective evaluations impacts the assumption made by existing design metrics, that software design can be quantified on an absolute, totally ordered scale. As a by-product of interviewing members of the software industry about design decisions we gained some insight on other important design issues as well. The second reason we examined recurring themes via GRC was validation. As we built our case study summaries, we coded our summaries and compared these codes to GRC and SRC. Finding matching codes between coded case summaries and GRC or SRC was an excellent vehicle for verifying our internal validity as we discuss in Section 5.

4.3. "Answers" to questions

Using RDM and NDM to guide the generalizing of our interview subjects' approaches to decision making [41,37], we interpreted each interview question as it related to the attributes of RDM and NDM. We found differences between RDM and NDM in the *goal, method, effect of environment,* and the *nature of the knowledge* employed in the decision as shown in Table 5. We address each of these in this order.

If a decision maker's goal was to *optimize* design (rational), then **information cues** were considered to indicate *right or wrong* decisions. If the decision maker's goal was to *satisfice* design (naturalistic), then **cues** were used only to indicate *better or worse* outcomes.

If a decision maker followed *consequential choice* (rational), then s/he **discussed numerous options** surrounding the decision to make a design change. If a decision maker followed *singular evaluation* (naturalistic), then s/he **did not discuss options**.

If a decision maker was *unconcerned about time pressure* and the external environment (rational), then s/he was **unconcerned with computational overhead** and **external goals**. On the other hand, if the decision maker was *concerned about time pressure* (naturalistic), then *dynamic conditions, real-time reactions, ill-defined tasks and goals* and*situation assessment* allowed external goals to influence decision making, thus providing **little time and point** in considering detailed computations.

If the decision maker was *cognizant of all possible courses of action* (rational), then *experience*, *knowledge* and *explicit searches* were used to reach decisions. If the decision maker was *not* cognizant of all possible courses of action (naturalistic), then s/he relied on general accumulation of experience or knowledge. Given these general interpretations, more detailed results illustrate similarities and differences within and across cases.

Using these interpretations with our specific relational coding, we classified the answers to each question of each interview as either NDM, RDM, N/A (when we did not have time to ask the question) or "?" (when the categorization was unclear). Table 6 shows the classifications for each question and the overall classification of the interview subject's approach to the decisions they discussed. The overall classification is based on the majority classification of the probing interview questions. Our results show that our interview subjects are primarily NDM-oriented. This

Table 5
Interpretations of interview questions with respect to decision making

Component	1 RDM	2 NDM
Decision goal	(1.1) *Optimizing:* Cues are right or wrong, quantifiable	(2.1) *Satisficing:* Cues are better or worse, not quantifiable
Decision method	(1.2) *Consequential choice:* Options are considered	(2.2) *Singular evaluation:* Options are not considered
Decision environment	(1.3) *Not concerned with computation overhead:* Time pressure is not a factor in decision making. External goals do not impact decision making. Cues are quantifiable	(2.3) *Dynamic conditions:* External goals impact decision making. Time pressure impacts decision making (2.4) *Real-time reactions:* Time pressure is an issue. Cues are from some trigger (2.5) *Ill-defined tasks & goals:* Externals impact a decision (2.6) *Situation assessment:* Cues are unquantifiable
Decision knowledge	(1.4) *Cognizant of all possible courses of action:* Specific experience based knowledge, explicit search of knowledge	(2.7) *Tacit based knowledge:* Accumulation of knowledge (2.8) *Experience-based knowledge:* Accumulation of experience

Table 6
Classifications of answers to interview questions, per interview

Interview ID		1*	2*	3	4	5*	6*	7*	8	9*	10	11*	12*	13	14	15	16	17	18	19	20	21	22	23	24	25
Decision	RDM		X				X							?	X		X	X	?		?		X	?		
	NDM	X		X	X	X		X	X	X	X	X	X			X				X		X			X	X
Cues	RDM												X	X	X			X			X	X			X	X
	NDM	X	X	X	X	X	X	X	X	X	X	X				X	X		X	X			X	X		
Knowledge	RDM		X	X			X		X								X	X					X	X	X	
	NDM	X			X	X		X		X	X	X	X	X	X	X			X	X	X	X				X
Options	RDM	X	X			X	X	X		X			X				X	X	X		X		X	X		X
	NDM			X	X				X		X	X		X	X	X				X		X			X	
Experience	RDM		X				X								X	X	X	X	X				X	X		
	NDM	X		X	X	X		X	X	X	X	X	X	X						X	X	X			X	X
Time pressure	RDM	X	X	X	X	X	N/A			X	X			X	N/A			X	X	X	X		X			
	NDM							X	X			X	X			X	X					X		X	X	X
External goals	RDM	N/A	X				N/A		X					X	X	X	N/A					X				
	NDM			X	X	X		X		X	X	X	X					X	X	X	X		X	X	X	X

* indicates Mentor perspective.

becomes apparent by examining the total number of categorizations in an RDM column (65) in Table 6 versus the total number of categorizations in an NDM column (101) in Table 6. When examining Mentor perspectives, we have 2 Mentor perspectives that were categorized as RDM or NDM and 6 categorized as NDM. When examining Developer perspectives that were categorized as RDM or NDM we have 4 categorized as RDM, 9 categorized as NDM and 4 that were not clear enough to categorize. We used the overall Decision question to report these numbers.

For Cues, Experience, Knowledge and External Goals, the interviewees' discussions of each was more aligned with the NDM interpretations (as provided in Table 6) than the RDM interpretations. For Alternatives the interview subjects' discussions were more aligned with the RDM interpretations (as provided in Table 5). For Time Pressure the interview subjects' discussions were almost evenly split between NDM and RDM. We list "N/A" where we did not have time to ask the question during the interview.

Two points are important to conclude from these results. First, we are not able to identify any pattern of decision making among interviews or between interview perspectives, solely from this categorization. Despite this, these results identify the foundations of decision making for the interview subjects' responses, and the application of our interpretations of NDM and RDM to the responses. This is particularly important as applying the interpretations in Table 5 to each case study was how we began our explanation building [66] for our decision making theory. The final step was to examine the context of each interview to determine the pattern that is not visible from these categorizations.

4.4. Case study summaries

We built case study summaries and compared them to each other to build an explanation of design decision making. We provide two case study summaries in Figs. 1, and 2 as examples of the 25 case studies we analyzed. We selected Developer NDM and Developer RDM to describe. Each case study shows evidence of three results that make up the three major components of the decision model we present as our final results. The first result is that software designers perform problem structuring when making decisions about software design. This reconfirms results found in [24]. The second result is that software designers incorporate information from their environment and their personal knowledge and experience when performing problem structuring. The third result is that a software designer's approach to decision making changes with respect to the structure of the problem about which they make design decisions. This last result emerges by comparing approaches to decision making across interviews, not only from looking at one interview.

Our description of the figures used for the case study summaries is as follows. We show problem structuring as a circle in our case studies. We use a circle because the structuring of one problem feeds right into the structuring of another problem [58]. Starting at the twelve o'clock position, marked with a dashed line, we move clockwise to the right, continuously structuring the problem as information is incorporated. We use a dashed line because the boundary between an ISP and a WSP is blurry [58]. The information used to structure the design problem is shown with quotes from each transcript, representative of the overall answer to a probing question from the CDM. The placement of this information with respect to the circle is arbitrary, but we have kept it consistent in our figures for readability. We highlight aspects of the interview, with respect to decision making, inside the structuring circle. Lastly, we provide an overview of the design change discussed by the interview subject, along the top of the figure.

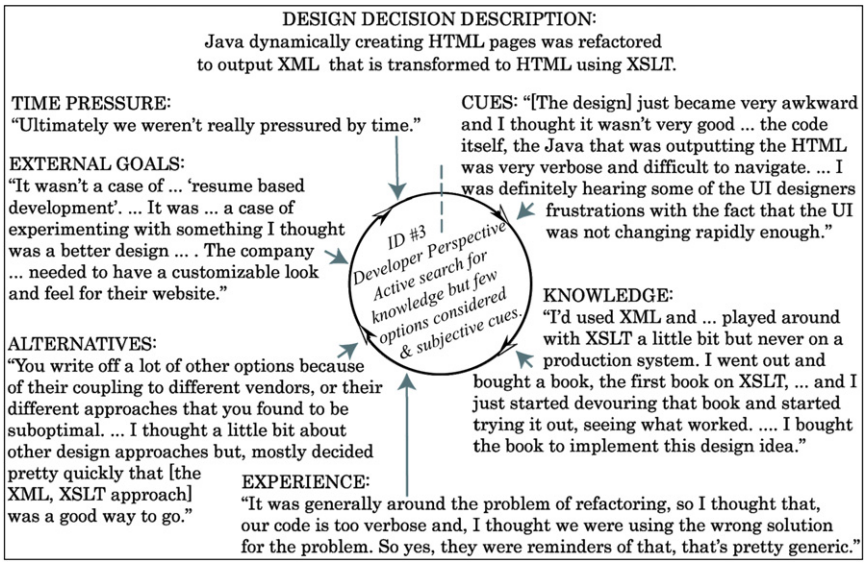

Fig. 1. Developer NDM case study summary.

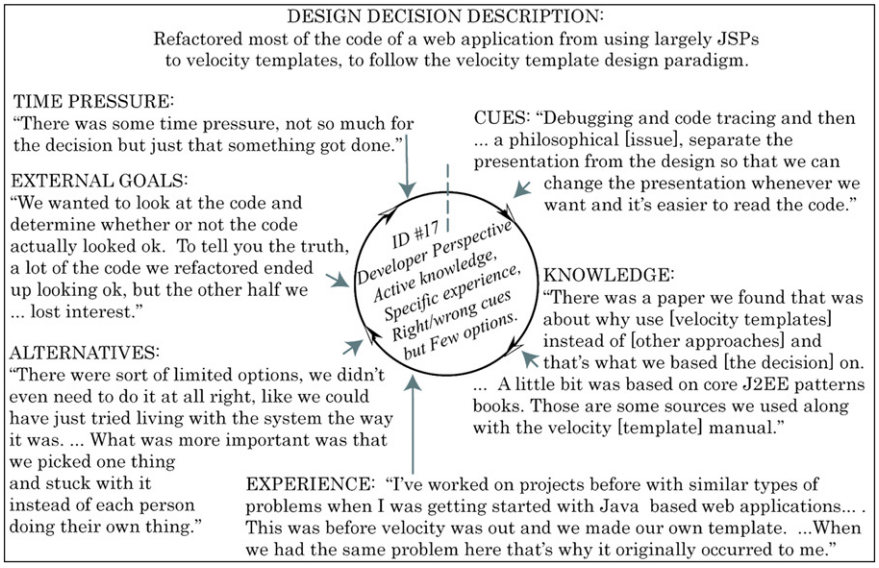

Fig. 2. Developer RDM case study summary.

4.4.1. Developer NDM, ID #3

Subject #3 was classified as taking a primarily naturalistic approach to design decisions, as shown in Table 6. We present this case in more detail here.

We begin with a quote from Interview Subject #3 to show support for problem structuring.

> *"I wasn't too happy with the design. I'd been talking to some friends and reading some articles about XSLT, and at that point it seemed like it had become mature enough to use. I had played with it earlier and it wasn't that mature but now it was mature, so I sold it to the managers, the idea of moving to this approach. I explained various business reasons why it would be important to do. Then I. . .built the first skeleton version. . ., a login page I changed to use XSLT. That was a nice cue that it worked. . ." ID #3 Developer*

Here the idea of problem structuring is not immediately clear, but the last two lines of this quote are important. Subject #3 had a negative feeling about the current state of a design (nb, did not say the current design was wrong) and had an idea about a design change. Without knowing if the design change would be successful, subject #3 prototyped a small version of the proposed design change and used that as a guide to the rest of the design change. This fits with the explanation of an ISP given that, "neither the guiding organization nor the attributes evoked from memory need at any time during the process provide a complete procedure nor complete information for designing. As a matter of fact, the entire procedure could conceivably be organized as a system of productions in which the elements already evoked from memory and the aspects of

design already arrived at up to a given point would serve as the stimuli to evoke the next set of elements'' [58]. Interview Subject #3 used the initial prototype as a stimulus to the rest of the design change and thus worked from an ISP to a WSP.

Again, problem structuring was accomplished by incorporating information from a designer's environment and by using knowledge and experience. We show highlights of the responses from subject #3 in Fig. 1 and incorporate additional quotes, where beneficial, as we describe each probe.

Cues. Indicators that a design change needed to occur were continuously mentioned by subject #3. Phrases such as ''feeling the pain'' and ''awkward'' and ''frustrated'' were common, all of which are difficult to quantify as ''wrong'', yet are clear indicators that the design is not right. These indicators came from the development environment and from Interview Subject #3. The quote in Fig. 1 clearly shows this (Table 5, #2.1).

Knowledge. The knowledge to implement the change came from an explicit search for a specific solution (Table 5, #1.4). This clear-cut approach to problem solving is very clearly represented in Fig. 1.

Experience. The experience used was generic, as stated verbatim, by subject #3 and shown in Fig. 1 (Table 5, #2.8).

Alternatives. Subject #3 did not consider options in solving the problem, as shown in Fig. 1, but had definitive reasons why the alternative chosen, was chosen (Table 5, #2.2).

External goals. The reference to ''selling'' the design change to managers is very suggestive of external goals that had to be addressed. Similarly a desire to experiment with a better idea, as shown in Fig. 1 impacted the design change (Table 5, #2.5).

Time pressure. Time pressure was not an issue in this design change and this is clearly represented in Fig. 1 (Table 5, #1.3).

We find subject #3 used a primarily naturalistic approach to a semi-structured design decision, but used facets of rational decision making when searching for a solution and acknowledging time pressure.

4.4.2. Developer RDM, ID #17

Subject #17 was classified as taking a primarily rational approach to design decisions, as shown in Table 6. We present this case in more detail here.

We begin with a quote from subject #17 to show support for problem structuring.

> *''It seems to me whenever you do refactoring, the hardest part is coming up with the plan, not the actual refactoring part, because once you have the plan, it's usually just the same thing over and over, like I'm going to be doing a find and replace or I'm going to look for code that looks like this and replace it with this.'' ID #17 Developer*

Subject #17 addresses, almost explicitly, the fundamental idea behind problem structuring. ''There is merit to the claim that much problem solving effort is directed at structuring problems, and only a fraction of it at solving problems once they are structured'' [58]. The quote by subject #17 states making a plan is harder than executing it, which we find quite similar to the idea that problem solving is more about structuring the problem rather than solving it once it is structured [58].

Again, problem structuring was accomplished by incorporating information from a designer's environment and by using knowledge and experience. We show highlights of the responses from subject #17 in Fig. 2 and incorporate additional quotes, where beneficial, as we describe each probe.

Cues. Subject #17 discusses subjective cues such as finding a web application difficult to test and debug, but the larger indicator that a design change needed to occur was not following a design heuristic. Separating the presentation layer from the business layer is a well-published concept in web applications [22]. Subject #17 examined code and found that it did not adhere to a popular heuristic (Table 5, #1.1).

Knowledge. The quote found in Fig. 2 is clear evidence of the explicit use of published references for solving the design problem (Table 5, #1.4).

Experience. Subject #17 heavily relied on specific past experiences to solve this problem (Table 5, #1.4). This is clear in the quote in Fig. 2.

Alternatives. Considering alternatives was done to a small degree, by subject #17. Initially we understood that no alternatives were considered, but the use of taglibs to solve the problem was a consideration, and pre-existing taglibs in the web application under development offered a measure of comparison to the chosen velocity template alternative (Table 5, #1.2).

External goals. Subject #17 identified a desire to keep the code clean, although from the quote in Fig. 2, we see this was not entirely executed (Table. 4.4, #2.5).

Time pressure. The design change was part of a project over the summer, but it did not have any serious time constraints, as per the quote in Fig. 2 (Table 5, #1.3).

We find subject #17 used a primarily rational approach to a well-structured design decision, and used extremely minimal facets of naturalistic decision making when identifying a design change needed to occur.

4.5. Cross case comparison

After completing the above case study summaries for each of the 25 interviews, we performed cross case comparison. We will use the above two cases as an example of the discussions we had to interpret our data.

Subject #3 had a relatively abstract design problem, in comparison to Interview subject #17. Subject #3 felt the

code he was working with was awkward or could be better, and heard other developers complaining about workflow issues. In contrast, subject #17 had an issue with separating presentation code from business code, a common heuristic in the design of web applications [22]. Here we see the cues to the design problem were fuzzier (ill-structured) for subject #3 than they were for subject #17. The problem definition was more precise for subject #17 than it was for Subject #3. While both interview subjects did an active search for knowledge and did not consider options to any large extent their use of past experiences was quite different. Subject #3 could not recall a specific past experience that helped in the discussed design problem while subject #17 used a specific past experience as a large indicator of the resolution to his design problem.

Our arguments continued in this fashion using our quantitative data, our interpretations of Table 5, and continuous reference to the transcripts. From this we generated enough of an understanding of each case to develop a decision model.

4.5.1. Problem structuring

Before describing the decision model, we take one last look at problem structuring, because it is an integral component of the decision model. In the above case studies, we describe the structure of the design change that each interview subject discussed. We cannot quantitatively define the structure of a problem, by the very definition of problem structuring. However, we can examine the structure of each problem relative to each other. Because the approach to decision making varied with respect to the structure of the problem as perceived by the interview subject, we show the structure of each design change discussed by the interview subjects, relative to each other. This is shown in Fig. 3. Two interview subjects (ID #18 and ID #25) discussed two distinct design changes during the interview, and both of these are shown in Fig. 3.

Again we show problem structuring as a circle, and a problem is more structured as it moves clockwise around the circle, beginning at the 12 o'clock position. We placed each design change on the circle according to the structure of the design problem discussed, and we found four divisions within our interviews, increasing in structure as we move clockwise around the circle. The first quadrant (upper right) primarily discussed design as idea generation. The second quadrant (lower right) primarily discussed large architectural changes to existing systems, based on developer instinct. The third quadrant (lower left) primarily discussed the use of design heuristics such as remove duplication, to recognize changes to an existing system. The fourth quadrant (upper left) discussed small code changes often as a result of customer need. We emphasize the blurry boundary between a WSP and an ISP and between cases within each quadrant.

4.6. Decision model

Given all of our results, we make three conclusions and present our decision model.

Conclusion #1. The more structured the design problem was, the more the interview subject primarily discussed rational approaches to solve the problem, and the more naturalistic approaches were facilitators to this discussion. In casual speak, the interview subject might have said "I could have tried this option, I could have tried that option, but I did not because my approach was right and it was right because of these constraints."

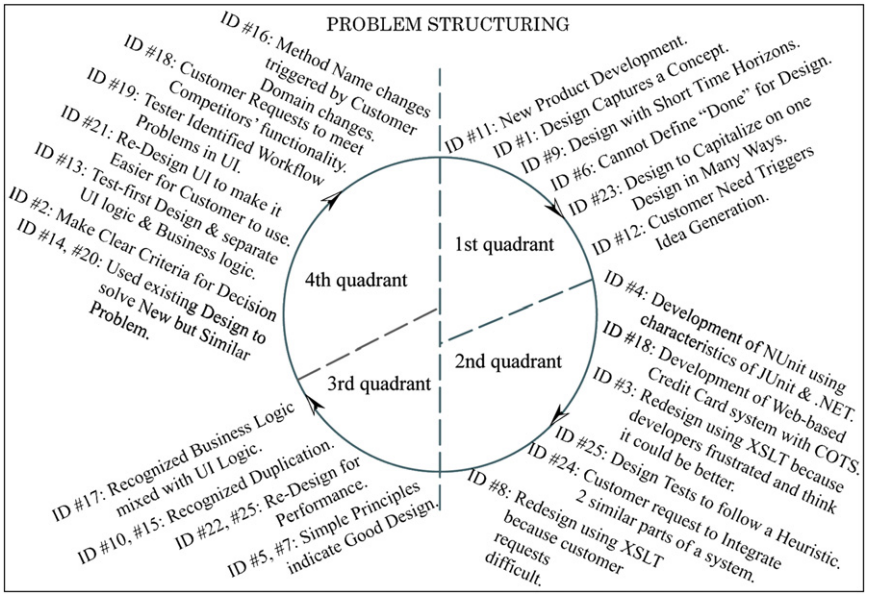

Fig. 3. Structure of design problems from interview subjects.

Conclusion #2 (the inverse of Conclusion #1). The less structured the design problem was, the more the interview subject primarily discussed naturalistic approaches to solving the design problem, and the more that rational approaches were facilitators to this discussion. In casual speak, the interview subject might have said, "I thought about this option, I thought about that option, but I only actually tried my approach because it seemed fitting and worked out okay."

To be clear, we never completely eliminated the use of RDM or NDM in any case study. There was always a combination of both. For example Interview Subject #17 followed RDM with respect to Cues, Knowledge, and Experience, but followed NDM with respect to Alternatives. Interview Subject #3 followed RDM with respect to Knowledge and Time but followed NDM with respect to Cues, Experience and Alternatives. From this we make a third conclusion.

Conclusion #3. When design decisions are made, either NDM or RDM is the dominant decision making approach, but aspects of the other decision making approach are used to implement the decision, depending upon the structure of the decision.

We used all three conclusions as a working theory while we analyzed all of our cases. We built on these conclusions to produce the following decision model.

The Design Decision Model that emerged from our results is shown in Fig. 4 as we describe three components of the model: Structuring Flow, Structuring Mechanisms and Structuring Perspectives. Decisions are made along the Structuring Flow as a result of incoming Structuring Mechanisms and using underlying Structural Perspectives.

4.6.1. Structuring flow

A consistent aspect that emerged from the interviews was the Structuring Flow. We define a Structuring Flow as the process of structuring an ill-structured software design problem to a well-structured software design problem. This result confirms existing design literature [24]. When software developers are faced with a design change they move from an idea of that change to a specific implementation of that change. Thus the circle in Fig. 4 is called the Structuring Flow and moves in a clockwise direction, starting from the "12" position. The product of one structuring is used in subsequent Structuring Flows, so the circle is continually exercised.

4.6.2. Structuring mechanisms

Another consistent aspect that emerged from the interviews is the use of numerous Structuring Mechanisms. We define a Structuring Mechanism as the information used by a software design decision maker to structure a problem and/or make a software design decision. All arrows pointing to the Structuring Flow in Fig. 4 are representative of Structuring Mechanisms. The bottom of Fig. 4 lists Structuring Mechanisms that can be used at any point along the Structuring Flow. The Structuring Mechanisms emerged from our data via our probing questions. Examples of Structuring Mechanisms are a software developer's personal experience, knowledge, ideas and opinions, their preferred evaluation criteria [24], the people with whom they interact, and existing models and work processes. All of these impact the way in which a design decision is structured.

4.6.3. Structuring perspectives

The last consistent aspect that emerged from the data is the perspective an interview subject takes in making a

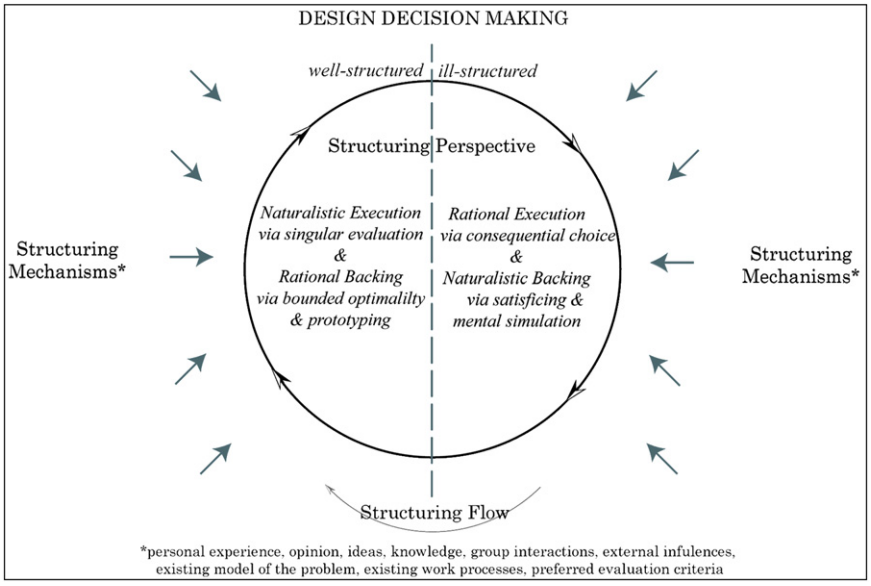

Fig. 4. Design decision model.

decision along the Structuring Flow. We define the Structuring Perspective as the underlying decision theory, one of rational or naturalistic decision making, that an interview subject uses in making a decision. In Fig. 4, the Structuring Perspectives are listed inside the circle created by the Structuring Flow. The definitions of RDM and NDM, are used to determine the Structuring Perspective. It is here that our results show a merger of two seemingly opposing decision approaches.

Our interviews show that the more ill-structured the design problem (e.g. requirements) the more the decision maker uses naturalistic approaches, via satisficing, but the more the decision maker uses rational approaches via consequential choice. The decision is naturalistic in that it uses mental modeling [1,4,24] and mental simulation [37,23] to subjectively evaluate advantages and disadvantages to a decision problem that has no right or optimal answer. The decision is rational in that it uses consequential choice [41,60] to compare decision alternatives.

Our interviews also show that the more well-structured the design problem (e.g. source code) the more the decision maker uses rational approaches via boundedly optimal solutions, but the more the decision maker uses naturalistic approaches via singular evaluation. The decision is rational in that a right or (bounded) optimal [60] decision is easily found because the problem is extremely focused (e.g. separating business logic from presentation code). The decision is naturalistic in that singular evaluation is used instead of consequential choice because the "right" alternative is found.

In other words, a decision in an ISP is rational in its execution with a foundation in naturalistic decision making and a decision in a WSP is naturalistic in its execution with a foundation in rational decision making. Our results show that software design decisions are made by integrating aspects of RDM and NDM, depending on how structured the design problem is perceived to be, by the decision maker. This perception is dependent upon his/her ability to access Structuring Mechanisms.

4.6.4. Summary

In the Design Decision Model above, a decision occurs anywhere along the Structuring Flow and the decision maker uses the corresponding Structuring Perspective to a greater or lesser extent, depending on the structure of the problem. The Structuring Perspective incorporates aspects of RDM and NDM and the extent to which aspects of these decision making approaches are incorporated depends upon the number and variety of Structuring Mechanisms a decision maker has access to, and utilizes. Lastly, the product of any structuring becomes the basis for future structuring. In Fig. 4 the top center of the circle created by the Structuring Flow is labeled with both "ill-structured" and "well structured" marking the extreme beginning and ending points, respectively, of any structuring. Decisions made become part of the structure for future decisions.

5. Validity

We discuss the construct, internal and external validity as well as a threat to the validity of our study.

5.1. Construct validity

The use of the CDM (Critical Decision Method) to examine decision making is a well-used tool [34,37]. The assumption, however, that a design change is a critical incident may be argued. We believe this to be a minor assumption because the purpose of the CDM is to examine *cognitive skills* used in decision making about a critical incident, not just the critical incident itself. Effecting design change is most certainly a cognitive skill requiring decision making [27,28,54].

5.2. Internal validity

We are able to internally validate our results in two ways. We validate our interpretations of the answers to the probing questions using our Specific Relational Coding. We validate our assigning of the decision making approach using our categorizations of the probing questions as NDM or RDM, as shown in Table 5.

5.2.1. Interpreting the probing questions

We compared our SRC to the summaries of each question, which we then coded. We summarized the responses of each question for each interview, in our own words, then coded that summary. We searched the SRC of each interview subject for matching codes, to verify our summary of the question. A match means that we found the codes from the coded summary in the codes generated by SRC, for each of the 7 questions asked. All of our case study comparisons showed at least 5 out of 7 matches between the coded summaries and the SRC of each interview. Typically when there was a mismatch, it was because the SRC found codes that did exist while the coded question summary showed what was not in the response to the interview question. For example, if time pressure was not an issue in a decision, the SRC contained other codes representative of what was an issue in a decision, whereas the coded summary contained the code !<<Time>>, to show that time was not an issue.

5.2.2. Assigning decision approaches

We compared the structure of the decision discussed to the decision making approach assigned to the decision in Table 6.

We found two interview subjects who were categorized as using NDM in their decision making, despite having a WSP. We found one interview subject who was categorized as using RDM in their decision making, despite having an ISP. Subject #19 was in a quality assurance role when the design change that was discussed occurred. We believe this gives a different perspective on a design problem and is

perhaps a factor in how a problem is structured. Subject #21 had the least experience of our interview subjects which is, as we have stated, a factor in how a problem is structured. We did not have enough time during the interview to ask subject #6 two of the 6 probing questions, so our categorization of this subject is weakened by a lack of information. While these are not the exact reasons for the inconsistency between the structure of the problem and the approach to decision making, they draw attention to an important issue: the background of the person designing the software has the potential to heavily impact the approach to design. This is beyond the scope of this work.

5.3. External validity

Given that we conducted case studies, not experiments, our external validity relies on generalization to theory (analytical generalization), and not statistical generalization. This approach to theoretical development is well-documented and employed in many social science disciplines [66]. All of our case studies align with our decision making model. For example, subject #3 found code "awkward" and "verbose" (qualitative cues) so he read up on XSLT as a solution to his problem (search for knowledge), "sold the idea" to business managers (external goals), and then prototyped his idea and fully implemented it once the prototype worked (singular evaluation of alternatives, satisficing). He felt little time pressure and was not reminded of specific past experiences.

5.4. Threats to validity

We address four topics that can be used as critiques of the validity of our study. The first is that most of the participants were familiar with, used or discussed agile software methods. It can be argued that the principles defined in the agile manifesto [2] align with the definition of naturalistic decision making [37]. As a result, our study may include a larger balance towards NDM versus RDM. To mitigate this, future work (already underway at the time of writing) involves observations of developers in non-agile environments to explore the NDM:RDM balance in these design environments.

The second issue is that our results mix Mentor and Developer perspectives, potentially impacting our results. We see this as potential for even further insight into design decision making. Part of our evaluation involved comparing Mentor and Developer perspectives to determine the similarities and differences between the two perspectives. While we saw some differences with respect to the use of knowledge (Mentor perspectives discussed active search more than Developer perspectives), there was similarity between the two perspectives.

The third issue is that we do not report the types of systems that were discussed in the design decision (expect to explain the context of the system). The primary reason for this is that the type of system was not the topic of analysis, individual design changes were. Because we do not have extreme variation in the types of systems discussed (e.g. personal projects and critical systems), because most of the systems were small to mid-size projects (because many were agile projects), and because we interviews designers about a design change not an end-to-end system, we see the issue of the type of system discussed to be a small threat to the validity of our study.

The last issue is that our results are subject to the weaknesses of retrospective interviews. Interview subjects reported their results as they remembered the design change and thus our results are subject to their recollection. In order to validate these results we will compare these results to the results of our observations in agile and non-agile environments to determine if what developers say they do matches what they actually do.

6. Conclusions

We presented results from a multi-case study of 25 software designers interviewed about design decisions they have made. Our results are quantitative and qualitative, occurring on numerous levels of abstraction, allowing us for multiple forms of internal validation. From these results we presented a model of software design decision making and make four important conclusions.

Conclusion #1 is that software design is primarily about problem structuring. Conclusion #2 is that the more structured the design problem was, the more the interview subject primarily discussed rational approaches to solve the problem, and the more that naturalistic approaches were facilitators to this discussion. Conclusion #3 is that the less structured the design problem was, the more the interview subject primarily discussed naturalistic approaches to solving the design problem, and the more that rational approaches were facilitators to this discussion. Conclusion #4 is that decision makers use either NDM or RDM as the dominant decision making approach but use aspects of the other decision making approach to implement their decision, depending upon the structure of the decision.

Our evidence shows that measures of design are most often qualitative, subjective and sometimes even based on gut feeling, which challenges the pursuit of quantitative and objective measures of design. Our evidence shows that software designers often use satisficing and singular evaluation in trying different approaches to design, which motivates the use of iterative design, which gives a designer more opportunity to approach a design in different ways. Lastly, our evidence shows that published knowledge in the form of textbooks and journals is not always used, which raises questions about the most effective methods to acquire design knowledge. Our empirical study impacts software engineering design metrics, processes and training by highlighting designers' inherent work processes in these areas.

Future work is directed at validating our decision model via observations at software companies (already underway

at the time of writing). Interviews are, by definition retrospective whereas observations are, by definition, current.

Acknowledgements

We thank all of our interview participants who took time from their work days to participate in our interviews.

References

[1] B. Adelson et al., The role of domain experience in soft design, IEEE Transactions on Software Engineering 11 (11) (1985).
[2] Agile Manifesto. Available from www.agilemanifesto.org (08/17/2005).
[3] Agile Conference. Available from www.xpuniverse.com (02/27/2006).
[4] S. Ahmed et al., Understanding differences between how novice & experienced designers approach design tasks, Research Engineering and Design 14 (2003) 1–11.
[5] J. Bach, The Challenge of Good Enough Software, American Programmer, October, 1995.
[7] L.R. Beach, Image theory: Personal & organizational decisions, in: G.A. Klein, J. Orasanu, R. Calderwood, C.E. Zsambok (Eds.), Decision Making in Action: Models and Methods, Ablex Publishing Corporation, Norwood, NJ, 1993.
[8] B. Brehmer, G.R.B. Joyce, Human Judgment: The SJT View, Elsevier Science Ltd, Amsterdam, 1988.
[9] L.C. Briand, S. Morasca, V.R. Vasili, Property-based software engineering measurement, IEEE Transactions on Software Engineering (1996) 22.
[10] L.C. Briand, Ch. Bunse, J.W. Daly, A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs, IEEE Transactions on Software Engineering 27 (2001) 6.
[11] B. Bruegge et al., Object-Oriented Software Engineering, Prentice Hall, New Jersey, 2004.
[12] C. Clegg, Psychology and information technology: The study of cognition in organizations, British Journal of Psychology 85 (1994) 449–477.
[13] J. Conklin, M. Begeman, gIBIS: A hypertext tool for exploratory policy discussion, ACM Transactions on Office Information Systems 6 (4) (1988) 303–331.
[14] R.W. Cooksey, Judgment Analysis: Theory, Methods and Applications, Academic Press, California, USA, 1995.
[15] B. Curtis et al., A field study of the soft. des. process for large systems, Communications of the ACM 31 (11) (1988).
[16] T. Demarco et al., Peopleware, second ed., Dorset House Pub. Co., New York, 1999.
[17] R.G. Dromey, A Model for Software Product Quality, IEEE Transactions on Software Engineering 21 (2) (1995).
[18] N. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, second ed., PWS Publishing Company; Cambridge University Press, 1996.
[19] J.C. Flanagan, The critical incident technique, Psychological Bulletin 51 (4) (1954).
[20] B. Flyvberg, Making Social Science Matter: Why Social Inquiry Fails and How it Can Succeed Again, Cambridge University Press, Cambridge, 2001.
[21] M. Fowler, Avoiding repetition, IEEE Software (2001).
[22] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object – Oriented Software, Addison-Wesley, Upper Saddle River, NJ, 1995.
[23] S. Gasson, Framing design: A social process view of information system development, in: Proceedings of the International Conference on Information Systems, Helsinki, Finland, 1998, pp. 224–236.
[24] R. Guindon, Designing the design process, HCI 5 (1990) 305–344.
[25] R. Guindon, Knowledge exploited by experts during software sys. design, International Journal of Man–Machine Studies 33 (1990) 279–304.
[26] J. Herbsleb, et al., Formulation and preliminary test of an empirical theory of coordination in soft. eng., in: Eur. Soft. Eng. Conf./ACM SIGSOFT Symp. Found. Soft. Eng; 2003.
[27] J. Highsmith, Agile Project Management, Addison Wesley, New Jersey, USA, 2004.
[28] J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, New Jersey, USA, 2003.
[29] D. Kahneman, A. Tversky, Prospect theory: An analysis of decision under risk, Econometrica 47 (2) (1979) 263–292.
[30] S. Kaner, with Lind L, Toldi C, Fisk S, Berger D; Facilitator's Guide to Participatory Decision Making; Sam Kaner, BC; 1996.
[31] L. Karsenty, An empirical evaluation of design rational documents, in: Proceedings of SIGCHI Conference on Human Factors in Computing Systems, Vancouver, BC, 1996, pp. 150–156.
[32] J. Kim, F.J. Lerch, Towards a model of cognitive process in logical design: comparing object-oriented and traditional functional decomposition software methodologies, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Monterey, CA, 1992, pp. 489–498.
[33] B. Kitchenham, S. Lawrence Pfleeger, N. Fenton, Towards a framework for software measurement validation, IEEE Transactions on Software Engineering 21 (1995) 12.
[34] G. Klein et al., Critical decision method for eliciting knowledge, IEEE Transactions on System, Man Cyber. 19 (1989) 3.
[35] M. Klein, Capturing design rationale in concurrent engineering teams, IEEE Computer 26 (1) (1993) 47–93.
[36] G.A. Klein, J. Orasanu, R. Calderwood, C.E. Zsambok (Eds.), Decision Making in Action: Models and Methods, Ablex Publishing Corporation, Norwood, NJ, 1993.
[37] G. Klein, Sources of Power, MIT Press, Cambridge, MA, 1998.
[38] Krippendorff, Content Analysis, V5, Sage Publications, London, 1980.
[39] J. Lee, K.-Y. Li, What's in design rationale? in: T.P. Moran, J.M. Carroll (Eds.), Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[40] R. Lipshitz, Decision making as argument-driven action, in: Klein, Orasanu, Calder-wood, Zsambok (Eds.), Decision Making in Action, Ablex Publishing Corporation, New Jersey, 1993.
[41] Luce et al., Games & Decisions, Wiley, New York, 1958.
[42] A. MacLean, R.M. Young, V.M.E. Bellotti, T.P. Moran, Questions, options, criteria: elements of design space analysis, in: T.P. Moran, J.M. Carroll (Eds.), Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[43] A. Malhotra et al., Cognitive processes in design, International Journal of Man–Machine Studies 12 (1980) 119–140.
[45] H. Montgomery, The search for a dominance structure in decision making: Examining the evidence, in: G.A. Klein, J. Orasanu, R. Calderwood, C.E. Zsambok (Eds.), Decision Making in Action: Models and Methods, Ablex Publishing Corporation, Norwood, NJ, 1993.
[46] D.A. Norman, Things that Make us Smart: Defending Human Attributes in the Age of the Machine, Addison-Wesley, New Jersey, USA, 1993.
[47] G.M. Olson, J.S. Olson, M. Storrosten, M. Carter, J. Herbsleb, H. Rueter, The structure of activity during design meetings, in: T.P. Moran, J.M. Carroll (Eds.), Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[48] J. Orasanu et al., The reinvention of decision making, in: Klein et al. (Eds.), Decision Making in Action, Ablex, New Jersey, USA, 1993.
[49] M.Q. Patton, Qualitative Research & Evaluation Methods, third ed., Sage Publications, California, USA, 2002.
[50] N. Pennington, R. Hastie, A theory of explanation-based decision making, in: G.A. Klein, J. Orasanu, R. Calderwood, C.E. Zsambok (Eds.), Decision Making in Action: Models and Methods, Ablex Publishing Corporation, Norwood, NJ, 1993.

[51] L.D. Phillips, A theory of requisite decision models, Acta Psychologica 56 (1984) 29–48.
[52] M. Poppendieck, T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison Wesley, Upper Saddle River, NJ, 2003.
[53] H. Rittel, M. Webber, Dilemmas in a general theory of planning, Policy Sciences 4 (1973) 155–169.
[54] S. Rugaber et al., Recognizing design decisions in programs, IEEE Software (1990).
[55] S.B. Shum, Analyzing the usability of a design rationale notation, in: T.P. Moran, J.M. Carroll (Eds.), Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[56] J. Siddal, Analytical Decision-Making in Engineering Design, Prentice Hall Inc., Englewood Cliffs, NJ, 1972.
[57] H. Simon, A behavioural model of rational choice, Quarterly Journal of Economics 69 (1) (1955) 99–118.
[58] H. Simon, The Structure of Ill Structured Problems, AI V4, 1973, pp. 181–201.
[59] S. Sonnetag, Expertise in professional software design, Journal of Applied Psychology 83 (5) (1998) 703–715.
[60] W.C. Stirling, Satisficing Games and Decision Making, with Applications to Engineering and Computer Science, Cambridge University Press, Cambridge, UK, 2003.
[61] Taylorism. Available from http://www.quality.org/TQM-MSI/taylor.html.
[62] A. Tversky, Elimination by aspects: A theory of choice, Psychological Review 79 (4) (1972) 281–299.
[63] A. Tversky, D. Kahneman, Judgment under uncertainty – heuristics and biases, in: D. Kahneman, P. Slovic, A. Tversky (Eds.), Judgment Under Uncertainty – Heuristics and Biases, Cambridge University Press, Cambridge, UK, 1982.
[64] K. Vicente, The Human Factor, Alfred A Knopf, Canada, 2003.
[65] D.B. Walz, et al., Inside a Software Design Team; Communication of the ACM, vol. 36, No. 10, October 1993.
[66] R.K. Yin, Case Study Research: Design & Methods, third ed., Sage Publications, California, USA, 2003.
[67] C. Zannier, et al., A qualitative empirical evaluation of design decisions, in: Workshop on Human & Social Factors of Soft. Eng., ACM Press, 2005.
[68] C. Zannier, pages.cpsc.ucalgary.ca/~zannierc/codes.html.